CORBA

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
  - erste Version 1991: CORBA 1.1
  - 1994: CORBA 2.0 (Interoperabilität: IIOP)
  - derzeit aktuell: CORBA 3.0 (aber: kaum vollst. konforme Produkte)

eine *Architektur*, kein Produkt!

- OMG (Object Management Group)
  - herstellerübergreifendes Konsortium (gegründet 1989)
  - Ziel: Bereitstellung von Konzepten für die Entwicklung verteilter Anwendungen mit objektorientierten Modellen
  - genauer: Definition und Entwicklung einer Architektur für kooperierende objektorientierte Softwarebausteine in verteilten heterogenen Systemen (→ "Middleware")

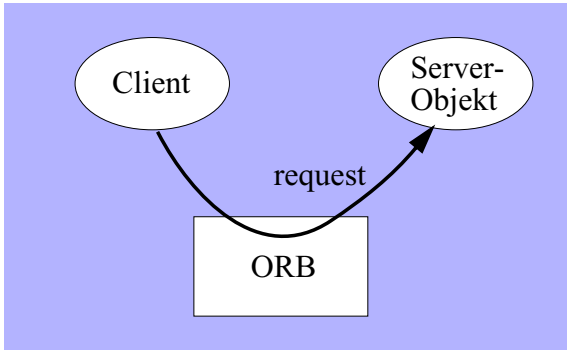
- CORBA beruht i.w. auf der Idee der Realisierung von Softwaresystemen aus interagierenden Komponenten
  + Services
  + Objektorientierung

Beachte: Objektorientierung selbst ist eigentlich ein "altes" Konzept

- Mitte der 1960er-Jahre (Programmiersprache "Simula")
- damals bereits fast alle Aspekte der Objektorientierung (Klassenhierarchien, virtuelle Klassen, Polymorphismus...)

CORBA - Übersicht

- *Objektmodell* (mit Aufrufsemantik etc.)
- *IDL* mit entspr. Generatoren und Compilern
- *Object Request Broker* (ORB) als Vermittlungsinfrastruktur

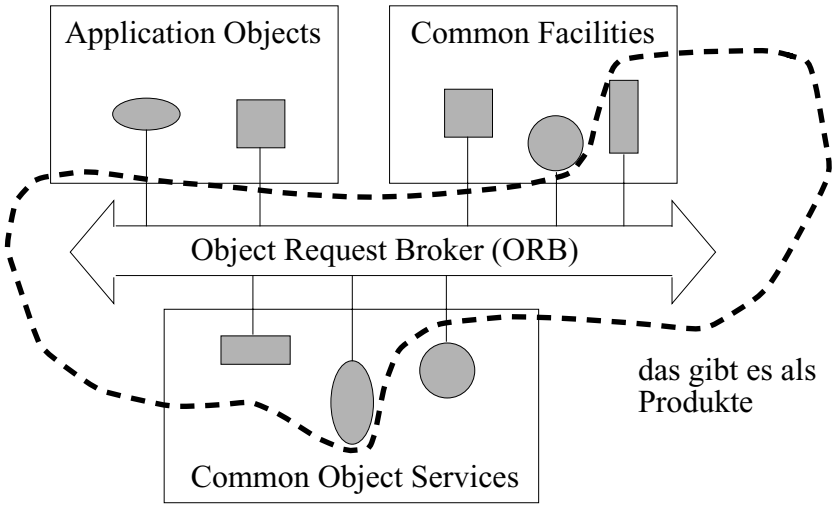


- *Systemfunktionen* in Form von Object Services
- Unterstützung von Anwendungen durch *Common Facilities* / Application Domains
- *Konventionen* bezüglich Schnittstellen, Protokollen etc.

- CORBA ist also eine *Infrastruktur* und unterstützt die Ausführung verteilter objektorientierter Systeme
- Die Entwurfs- und Spezifikationsphase solcher Systeme werden mit anderen Konzepten unterstützt, z.B. *UML* ("Unified Modeling Language"), mit der u.a. Diagrammnotationen standardisiert werden

Object Management Architecture

- "OMA" ist die *Referenzarchitektur* der OMG, die die wesentlichen Bestandteile einer Plattform für verteilte objektorientierte Applikationen definiert



- *Application Objects*: Objekte der eigentl. Anwendung
  - gehören damit nicht zur CORBA-Infrastruktur
- *ORB*: Vermittlung zwischen verschiedenen Objekten; Weiterleitung von Methodenaufrufen etc.
  → Ortstransparenz, Kommunikation...
- *Object Services*: objektorientierte Schnittstelle zu standardisierten wichtigen Diensten
- *Common Facilities*: allgemein nützliche Dienste
  - nicht notwendigerweise Teil aller CORBA-Implementierungen

Object Services (1)

- Basisdienste als systemweite Infrastruktur
  - mit objektorientierter Schnittstelle
- COSS (Common Object Services Specification)
  - Realisierung ist für voll CORBA-konforme Produkte verpflichtend
  - nicht alle Services sind aber vollständig spezifiziert / realisiert!

---

1) Ereignismeldung
   - Weiterleitung asynchroner Ereignisse an Ereigniskonsumenten
   - Einrichten von "event channels" mit Operationen wie push, pull...
2) Persistenz
   - Dauerhaftes Speichern von Objekten auf externen Medien
3) Naming
   - Erzeugung von Namensräumen
   - Abbildung von Namen auf Objektreferenzen
   - Lokalisierung von Objekten
4) Lifecycle
   - Erzeugen, Löschen, Kopieren, Verlagern,... von Objekten
5) Concurrency
   - Semaphore, Locks,...
6) Externalization
   - Export von Objekten in "normale" Dateien

Object Services (2)

7) Transactions
   - 2-Phasen-Commit etc.

8) Time
   - Uhrensynchronisation etc.

9) Security

10) Licensing
   - Management von Lizenzdiensten für Komponenten

11) Trading
   - Matching von Services zu einer Service-Beschreibung eines Clients

12) Replikation
   - Sicherstellung der Konsistenz replizierter Objekte in einer verteilten Umgebung

- es gibt noch einige weitere Services...

Common Facilities

- Höherwertige Dienste für ein breites Spektrum von Anwendungsbereichen

- Bereitstellung allgemein interessanter Funktionalität
  - analog zu grossen Klassenbibliotheken

- Horizontal ”Common Facilities”
  - Basisfunktionalität, die für verschiedene Anwendungsbereiche von Nutzen ist
    - secure time
    - internationalization (“...will enable developers to use an application in their own language using their own cultural conventions...will allow the developer to use a culture’s numeric and currency conventions...)
    - user interface
    - information management (z.B. Speicherung komplexer Objektstrukturen, Formatkonvertierungen,...)
    - systems management (z.B. Installation, Konfiguration... von Objekten)
    - task management (Workflow, lange Transaktionen...)

- Vertikale Common Facilities (“Domain Facilities”)
  - Basisfunktionalität für diverse Marktsegmente, z.B. Banken, Finanzdienste, Gesundheitswesen...
  - vgl. “application frameworks”, “business objects” etc.

OMA Component Definitions

Object Request Broker - commercially known as CORBA, the ORB is the communications heart of the standard. It provides an infrastructure allowing objects to converse, independent of the specific platforms and techniques used to implement the objects. Compliance with the Object Request Broker standard guarantees portability and interoperability of objects over a network of heterogeneous systems.

Object Services - these components standardize the life-cycle management of objects. Interfaces are provided to create objects, to control access to objects, to keep track of relocated objects, and to control the relationship between styles of objects (class management). Also provided are the generic environments in which single objects can perform their tasks. Object Services provide for application consistency and help to increase programmer productivity.

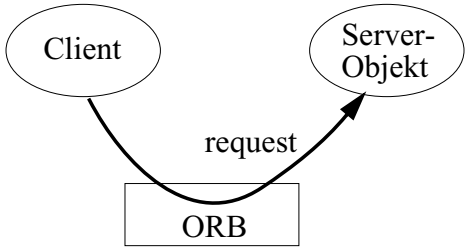
Common Facilities - Common Facilities provide a set of generic application functions that can be configured to the specific requirements of a particular configuration. These are facilities that sit closer to the user, such as printing, document management, database, and electronic mail facilities. Standardization leads to uniformity in generic operations and to better options for end users for configuring their working environments.

Domain Interfaces - Domain Interfaces represent vertical areas that provide functionality of direct interest to end-users in particular application domains. Domain interfaces may combine some common facilities and object services, but are designed to perform particular tasks for users within a certain vertical market or industry.

Quelle: http://www.omg.org/omaov.htm

Kommunikation zwischen Objekten

- Menge interagierender Objekte typw. in zwei Rollen
  - Client-Objekt (Aufrufer)
  - Server-Objekt



- Methodenaufruf durch requests unterschiedl. Semantik
  - synchron (insbes. bei Rückgabewerten; analog zu RPC)
  - “verzögert synchron” (Aufrufer wartet nicht auf das Ergebnis sondern holt es sich später ab)
  - “one way” (asynchron: Aufrufer wartet nicht)

- Beim Methodenaufruf muss angegeben werden
  - das Zielobjekt
  - die Parameter
  - ggf. Angaben über Exceptions und Rückgabewerte

Interface Description Language (IDL)

- Sprache zur Definition von Schnittstellen (Parameter, Attribute, Superklasse bzgl. Vererbung, Exceptions...)
- lexikalisch an C++ angelehnt
- Bsp: oneway void move (in long x, in long y)

Eingabe-, Rückgabe- oder Eingabe- und Rückgabeparameter

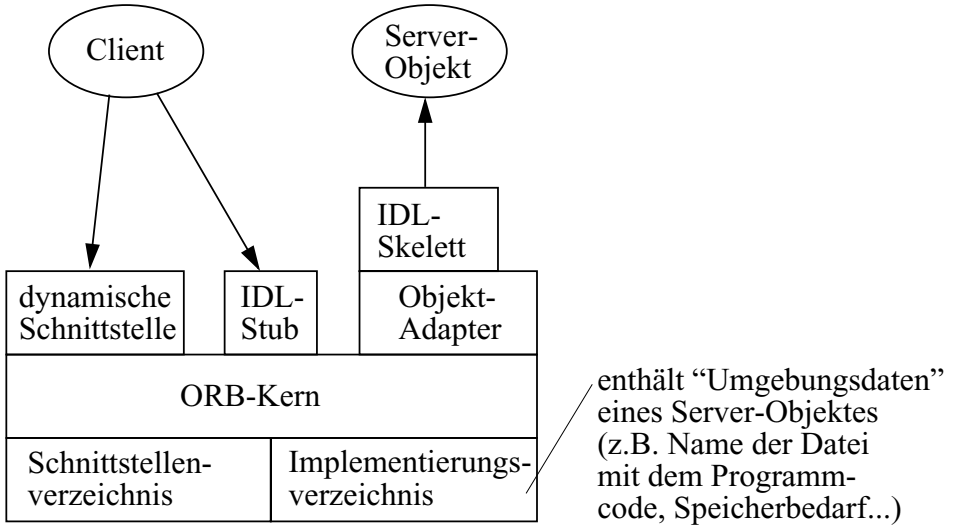
- grundsätzlicher Aufbau:

[oneway] <op_type_spec> <identifier> ($param_1$,....,$param_L$)
[raises($except_1$,...,$except_N$)] [context($name_1$,...,$name_M$)]

erlaubt Angabe möglicher Exceptions, die bei Durchführung der Operation auftreten können

operationsspezifische Informationen, die dem Operationsaufruf mitgegeben werden können

ORB



- ORB bietet einem Client zwei Arten von Schnittstellen für den Methodenaufruf an
  - statische Schnittstelle (Erzeugung von Stubs aus der IDL-Beschreibung analog zu RPCs)
  - dynamische Schnittstelle (Client kann zur Laufzeit das Schnittstellenverzeichnis abfragen und einen geeigneten Methodenaufruf zusammenstellen)
- Objektadapter: Steuert anwendungsunabhängige Funktionen des Server-Objekts
  - z.B. Aktivierung des Server-Objektes bei Eintreffen eines requests, Authentifizierung von requests, Zuordung von Objektreferenzen zu Objektinstanzen etc.
  - zuständig ausserdem für Registrierung von Services
  - es gibt einen standardisierten Basic Object Adapter (BOA), der für viele Anwendungen ausreichende Grundfunktionalität bereitstellt

Server-Objekte

- Bereitsstellung von Services (analog zu Prozeduren, die im Rahmen von RPCs verwendet werden)

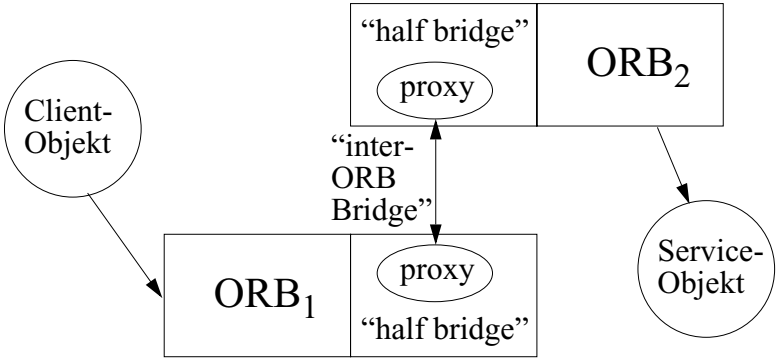
- Objekte können ein aus der IDL-Spezifikation generiertes Objekt-Skelett nutzen

- Objekt muss sich beim lokalen Objekt-Adapter anmelden und dabei eine “server policy” angeben
  - *Shared Server:* kann mit mehreren anderen aktiven Server-Objekten von einem einzigen Prozess verwaltet werden
  - *Unshared Server*
  - *Server per Method:* Start eines eigenen Prozesses bei Methodenaufruf
  - *Persistent Server:* ein Shared Server, der von CORBA initial bereits gestartet wurde

- Objekt muss sich ferner beim Implementierungs-verzeichnis anmelden
  - damit es bei einem Methodenaufruf gefunden wird

CORBA 2.0

- Insbesondere Interoperabilität von ORBs verschiedener Herstellerimplementierungen gefordert



- ORB Bridge: Formatkonvertierung und Weiterleitung eines requests etc. an einen anderen ORB
  - Schnittstellen und Konventionen für solche Bridges sind im CORBA-Standard festgelegt
  - Bridge besteht aus zwei Teilen mit einer CORBA-Schnittstelle, welche bei Bedarf Proxy-Objekte erzeugen, die die Aufrufkonvertierung vornehmen

- Inter-ORB-Kommunikation mittels bestimmter Protokolle
  - z.B. GIOP (General Inter-ORB Protocol): aufTCP/IP-aufbauende Realisierung; im “Internet Inter-ORB Protocol” (IIOP) festgelegt

CORBA - weitere Entwicklungen (insbesondere CORBA 3.0)

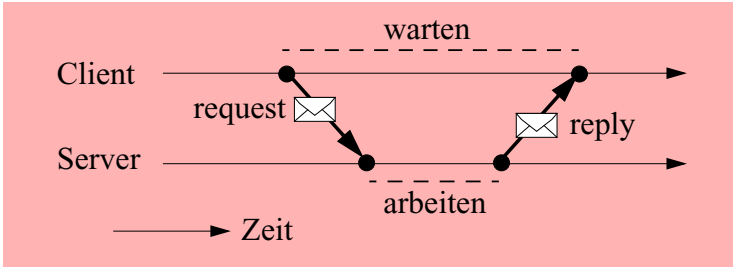
- Messaging Service
  - Objekte können sich asynchrone Nachrichten schicken (store & forward)
- Objects by Value
  - Kopie eines Objektes wird übergeben, nicht nur Referenz
- Persistente Objekte
  - "Abspeichern" von Objekten
- Komponenten-Modell
- Java-Unterstützung
  - Generieren von IDL aus Java bzw. Java-RMI ("reverse mapping")
- Firewall-Unterstützung
  - klassische Firewalltechnik (z.B. Services identifiziert mit Portnummern) versagt teilweise; Callbacks erscheinen als Aufruf von aussen...
- Minimum CORBA
  - Unterstützung von embedded systems (i.w. Weglassen von Dynamik)
- Realzeit-Unterstützung
- Fault Tolerant CORBA
  - durch redundante Einheiten

hierzu gibt es z.Z. wenig mehr als gutgemeinte Absichten (bestenfalls Experimentelles)

- Interoperable Naming Service mit "CORBA-URLs"
  - z.B.: iioploc://meinefirma.com:683/NamingService
  - löst damit das Bootstrapping-Problem des Namensdienstes

Neu: Messaging Service

- Asynchrones Kommunikationsparadigma
- Motivation:
  - mobile Geräte (PDA, Laptop,...) sind oft nicht online
  - bei sehr grossen verteilten Systemen sind unausweichlich stets einige Geräte bzw. Services nicht erreichbar (Netzprobleme etc.)
- CORBA basierte bisher auf einer engen ("synchronen") Kopplung von Client und Server



- Asynchron:
  - Entkopplung von Sender / Empfänger
  - Sender blockiert nicht solange bis Nachricht angekommen ist
  - Nachricht kann von Hilfsinstanzen ("Router") zwischengespeichert werden, bis Empfänger (bzw. bei Antwort: Sender) erreichbar ist
  - Antwort kann ggf. von anderem Client als ursprünglichem Sender entgegengenommen werde

Asynchronous Method Invocation

- Bisherige Möglichkeiten eines Methodenaufrufs in CORBA:
  - "synchron" (insbes. bei Rückgabewerten; analog zu RPC)
  - "verzögert synchron" (Aufrufer wartet nicht auf das Ergebnis, sondern holt es sich später ab)
  - "one way" (Aufrufer wartet nicht) mit "best effort"-Semantik ("fire and forget")

gedacht war an UDP-Implementierung; Semantik (z.B. Fehlermeldung bei Misslingen?) implementierungsabhängig

---

- Neu: Asynchronous Method Invocation (AMI)
  - bisher eher umständlich mit mehreren Threads simuliert
- Zwei Aufruftechniken bei AMI:
  - (1) Callback
    - Client gibt dem Aufruf eine Objektreferenz für die Antwort mit
    - Callback-Objekt kann sich im Prinzip irgendwo befinden
    - Kommunikations-Exceptions werden im Callback-Objekt ausgelöst
  - (2) Polling
    - Client erhält sofort ein Objekt zurück, das er für Polling oder zum Warten auf Antwort nutzen kann

Time-independent Invocation (TII)

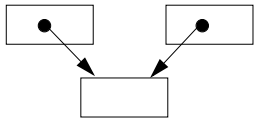
- Teil des Messaging Services: Aufruf von Objekten, die nicht aktiv sind oder zeitweise nicht erreichbar sind
- Aufruf-Nachrichten werden von zwischengeschalteten "Router Agents" verwaltet
  - Store and Forward-Prinzip
  - Router Agent beim Client ermöglicht disconnected operations
  - Router Agent beim Server kann dessen Eingangsqueue verwalten
- "Interoperable Routing Protocol" sorgt dafür, dass Router Agents verschiedener Hersteller interagieren
- Quality of Service (QoS) steuerbar (als "Policy")
  - z.B. max. Round Trip-Zeit: dadurch müssen Router Agents Nachrichten nicht beliebig lange aufbewahren
  - oder z.B. Aufrufreihenfolge: Soll Router seine gespeicherten Aufträge zeitlich geordnet oder nach Prioritäten oder... ausliefern?

- Beachte: QoS ist ein gewichtiger Name für ein einfaches Prinzip ohne feste Garantien

Objects by Value

- Bisher war nur *Referenzübergabe* möglich
  - um es Objekten zu gestatten, Methoden anderer Objekte aufzurufen, konnten bisher Objektreferenzen als Parameter übergeben werden
  - Objekt selbst bleibt aber "am Platz", Aufruf wird also immer als Fernaufruf über das Netz geschickt
  - ferner kommt es zum gelegentlich unerwünschten Aliasing-Effekt: unbedachte Rückwirkungen auf das "Originalobjekt"

- Bei *Wertübergabe* wird das Objekt serialisiert und im Adressraum des Empfängers eine Kopie angelegt
  - Marshalling des Objektzustandes (d.h. der *Daten*)
  - auf Empfängerseite existiert eine Factory, die das Objekt als Kopie (mit eigener Identität) erzeugt
  - was geschieht bei Alias-Zeigern bzw. Zyklen bei der Serialisierung komplexer Strukturen?



- Bei heterogenen Umgebungen: Wie transportiert man das *Verhalten* des Objektes zum Empfänger?
  - es handelt sich um ausführbaren Code (für welche Plattform?)
  - kann von verschiedenen Sprachen (C, Java,...) erzeugt worden sein
  - einfach bei Java auf beiden Seiten: Bytecode ist unabhängig vom Maschinentyp durch die JVM in gleicher Weise interpretierbar
  - ansonsten muss sich die Factory beim Empfänger den Code besorgen (aus lokaler Bibliothek, übers Netz...)

- Leider können jedoch keine normalen CORBA-Objekte per Value übergeben werden, nur sogen. "valuetypes"
  - neues Konstrukt der IDL (→ für Anwender dadurch kompliziert)
  - Wertübergabe bel. Objekte wäre zu schwierig

Real-Time CORBA

- Ziel: Vorhersagbares Ende-zu-Ende-Verhalten (insbesondere beim entfernten Methodenaufruf)
  - sowohl für "Hard Real-Time"
  - als auch für "Soft Real-Time" (mit nur statistischen Aussagen)

- Basiert auf einem zugrundeliegendem Realzeit-BS

- Einige Mechanismen:
  - Prioritäten
  - Timeouts für Aufrufe
  - Multithreading (mit geeignetem Scheduling), Threadpools
  - private (statt gemultiplexte) Verbindungen
  - austauschbare Kommunikationsprotokolle

- Anwendungsbereiche z.B.:
  - verteilter Telefonswitch
  - Prozessautomatisierung
  - Avionics
  - ...

CORBA und Java

- Java ist weit verbreitet (als "Internetprogrammier-sprache") und in gewissem Sinn eine "Konkurrenz"
  - allerdings hegemonistisch im Sinne von "überall Java / JVM"

- Ziel: Interoperabilität durch Zusammenführung von Java RMI und CORBA IIOP

  - Remote Method Invocation: Entfernter Methodenaufruf mit Transport (und dabei Serialisierung) auch komplexer Objekte
  - auch Code (Bytecode) wird übertragen und remote ausgeführt
  - damit kann z.B. ein Proxy eines Servers (z.B. ein Druckertreiber) jeweils aktuell zur Laufzeit vom Server zum Client geschickt werden
  - der Proxy kann dann (unsichtbar für den Client) z.B. mit dem Server ein privates Protokoll realisieren oder sonstige Dinge lokal tun

- IDL automatisch aus Java-Programmen generieren
  - "reverse mapping" (weiterhin existiert natürlich IDL → Java mapping)
  - Java-Programmierer brauchen kein IDL zu nutzen und zu lernen
  - aus Java heraus sind so Objekte anderer Sprachen ansprechbar (bzw. z.B. Java-Server, der von Clients anderer Sprachen genutzt werden kann)

- Aufbrechen des "single language Paradigmas" von Java
  - Java-Objekte werden von CORBA aus zugreifbar

- Kleinere Einschränkungen jedoch notwendig
  - Java-RMI und CORBA-IDL sind nicht deckungsgleich